PERANCANGAN DAN PENGUJIAN *LOAD BALANCING* DAN *FAILOVER* MENGGUNAKAN NGINX

Disusun sebagai salah satu syarat menyelesaikan Program Studi Strata I pada Jurusan Informatika Fakultas Komunikasi dan Informatika

Oleh:

RAHMAD DANI

L 200 130 120

PROGRAM STUDI INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

2017

HALAMAN PERSETUJUAN

PERANCANGAN DAN PENGUJIAN *LOAD BALANCING* DAN *FAILOVER* MENGGUNAKAN NGINX

PUBLIKASI ILMIAH

oleh:

RAHMAD DANI

L 200 130 120

Telah diperiksa dan disetujui untuk diuji oleh:

Dosen Pembimbing

Fajar Suryawan, S.T., M.Eng.Sc, PhD

NIK.924

HALAMAN PENGESAHAN

PERANCANGAN DAN PENGUJIAN *LOAD BALANCING* DAN *FAILOVER* MENGGUNAKAN NGINX

OLEH

RAHMAD DANI

L 200 130 120

Telah dipertahankan di depan Dewan Penguji
Fakultas Komunikasi dan Informatika
Universitas Muhammadiyah Surakarta
Pada hari Sabtu, 21 Januari 2017
dan dinyatakan telah memenuhi syarat

Dewan Penguji:

1. **Fajar Suryawan, S.T., M.Eng.Sc, PhD.** (.....)
   (Ketua Dewan Penguji)
2. **Dr. Heru Supriyono, M.Sc.** (.....)
   (Anggota I Dewan Penguji)
3. **Dr. Ir. Bana Handaga, M.T.** (.....)
   (Anggota II Dewan Penguji)

Publikasi ilmiah ini telah diterima sebagai salah satu persyaratan
Untuk memperoleh gelar sarjana
Tanggal 2 - 2 - 2017

Mengetahui,

Dekan
Fakultas Komunikasi dan Informatika

Husni Thamrin, S.T., M.T., Ph.D.
NIK : 706

Ketua Program Studi
Informatika

Dr. Heru Supriyono, M.Sc.
NIK:970

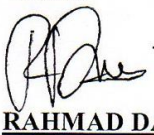
PERNYATAAN

Dengan ini saya menyatakan bahwa dalam skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaan di suatu perguruan tinggi dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan orang lain, kecuali secara tertulis diacu dalam naskah dan disebutkan dalam daftar pustaka.

Apabila kelak terbukti ada ketidakbenaran dalam pernyataan saya di atas, maka akan saya pertanggungjawabkan sepenuhnya.

Surakarta, 29 Januari **2017**

Penulis



RAHMAD DANI

L 200 130 120

UNIVERSITAS MUHAMMADIYAH SURAKARTA
FAKULTAS KOMUNIKASI DAN INFORMATIKA
PROGRAM STUDI INFORMATIKA
Jl. A Yani Tromol Pos 1 Pabelan Kartasura Telp. (0271)717417, 719483 Fax (0271) 714448
Surakarta 57102 Indonesia. Web: http://informatika.ums.ac.id. Email: informatika@ums.ac.id

SURAT KETERANGAN LULUS PLAGIASI

012/A.3-II.3/INF-FKI/I/2017

Assalamu'alaikum Wr. Wb

Biro Skripsi Program Studi Informatika menerangkan bahwa :

Nama	:	RAHMAD DANI
NIM	:	L200130120
Judul	:	PERANCANGAN DAN PENGUJIAN LOAD BALANCING DAN FAILOVER MENGGUNAKAN NGINX
Program Studi	:	Informatika
Status	:	**Lulus**

Adalah benar-benar sudah lulus pengecekan plagiasi dari Naskah Publikasi Skripsi, dengan menggunakan aplikasi Turnitin.

Demikian surat keterangan ini dibuat agar dipergunakan sebagaimana mestinya.

Wassalamu'alaikum Wr. Wb

Surakarta, 24 Januari 2017

Biro Skripsi Informatika

Ihsan Cahyo Utomo, S.Kom., M.Kom.


wisuda 2017
wisuda maret - DUE 17-Jan-2017
Roadmap
Paper 2 of 20
Originality
GradeMark
PeerMark
PERANCANGAN DAN PENGUJIAN LOAD BALANCING DAN FAILOVER
BY RAHMAD DANI
turnitin
5%
SIMILAR
--
OUT OF 0
PERANCANGAN DAN PENGUJIAN *LOAD BALANCING* DAN *FAILOVER* MENGGUNAKAN NGINX
Abstrak
Situs web dengan traffic yang tinggi dapat memiliki beban kerja yang berat pada sisi server, yang dapat mengakibatkan performa server menurun. Hal ini juga dapat menyebabkan kegagalan sistem secara keseluruhan. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan menerapkan teknik *load balancing* dan failover. *Load balancing* merupakan teknologi untuk melakukan pembagian beban kepada beberapa server, memastikan tidak terjadi kelebihan beban pada salah satu server. Sementara itu, Failover merupakan kemampuan suatu sistem untuk berpindah ke sistem cadangan jika sistem utama mengalami kegagalan. Dalam penelitian ini *load balancing* dengan teknik failover akan diimplementasikan pada sistem operasi Ubuntu Software inti yang digunakan dalam penelitian ini adalah Nginx dan KeepAlived. Nginx akan berfungsi sebagai *load balancer*, sedangkan KeepAlived untuk mengimplementasikan teknik failover. Beberapa skenario telah disiapkan untuk menguji sistem *load balancing* yang telah dirancang. Pengujian dilakukan dengan menggunakan Jmeter. Berdasarkan pengujian yang telah dilakukan, sistem yang dirancang berhasil membagikan beban permintaan dan dapat terus bekerja walaupun terjadi kegagalan pada server *load balancer* ataupun kegagalan pada server backend. Selain itu, dalam beberapa pengujian, dengan menggunakan *load balancing* terbukti mampu menurunkan waktu respon dan meningkatkan thoughput pada sistem sehingga mampu meningkatkan performa sistem. Mengacu pada hasil penelitian ini, sistem *load balancing* dan failover menggunakan nginx dapat dijadikan salah satu solusi pada sistem web server dengan situs web yang memiliki traffic tinggi.
Kata Kunci: load balancing, jaringan, nginx, keepalived.
Abstract
High-traffic websites can have heavy workload on the server side, which causes a decrease in the server performance. It might also lead to system failures. A distributed system using load balancing with failover is one of the solutions to overcome this problem. Load balancing is a technology to divide workload among multiple servers, ensuring that no one server is overloaded. Failover is the ability to switch to a secondary system on the event of failure of the primary system. In this
PAGE: 1 OF 15
Match Overview
1 eprints.ums.ac.id Internet source 1%
2 etd.eprints.ums.ac.id Internet source <1%
3 www.bizrate.com Internet source <1%
4 cahguci.blogspot.com Internet source <1%
5 fki.ums.ac.id Internet source <1%
6 cs.ulb.ac.be Internet source <1%
7 Submitted to Universid... Student paper <1%
8 Kosowski, Adrian, and ... Publication <1%
Text-Only Report

PERANCANGAN DAN PENGUJIAN *LOAD BALANCING* DAN *FAILOVER* MENGGUNAKAN NGINX

Abstrak

Situs web dengan *traffic* yang tinggi dapat memiliki beban kerja yang berat pada sisi server, yang dapat mengakibatkan performa server menurun. Hal ini juga dapat menyebabkan kegagalan sistem secara keseluruhan. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan menerapkan teknik *load balancing* dan *failover*. *Load balancing* merupakan teknologi untuk melakukan pembagian beban kepada beberapa server, memastikan tidak terjadi kelebihan beban pada salah satu server. Sementara itu, *Failover* merupakan kemampuan suatu sistem untuk berpindah ke sistem cadangan jika sistem utama mengalami kegagalan. Dalam penelitian ini *load balancing* dengan teknik *failover* akan diimplementasikan pada sistem operasi Ubuntu. Software inti yang digunakan dalam penelitian ini adalah Nginx dan KeepAlived. Nginx akan berfungsi sebagai *load balancer*, sedangkan KeepAlived untuk mengimplementasikan teknik *failover*. Beberapa skenario telah disiapkan untuk menguji sistem *load balancing* yang telah dirancang. Pengujian dilakukan dengan menggunakan Jmeter. Berdasarkan pengujian yang telah dilakukan, sistem yang dirancang berhasil membagikan beban permintaan dan dapat terus bekerja walaupun terjadi kegagalan pada server *load balancer* ataupun kegagalan pada server backend. Selain itu, dalam beberapa pengujian, dengan menggunakan *load balancing* terbukti mampu menurunkan waktu respon dan meningkatkan *throughput* pada sistem sehingga mampu meningkatkan performa sistem. Mengacu pada hasil penelitian ini, sistem *load balancing* dan *failover* menggunakan nginx dapat dijadikan salah satu solusi pada sistem web server dengan situs web yang memiliki *traffic* tinggi.

Kata Kunci: load balancing, jaringan, nginx, keepalived.

Abstract

High-traffic websites can have heavy workload on the server side, which causes a decrease in the server performance. It might also lead to system failures. A distributed system using load balancing with failover is one of the solutions to overcome this problem. Load balancing is a technology to divide workload among multiple servers, ensuring that no one server is overloaded. Failover is the ability to switch to a secondary system on the event of failure of the primary system. In this research, Nginx and KeepAlived are the key software packages. Nginx will be used as the load balancer, while KeepAlived will be used to configure the failover. Both methods will be implemented over the Ubuntu operating system. A number of scenarios were designed to test the designed load balancing system. Test will be conducted using JMeter software package. Based on the result of the test, the designed system was able to divided request workload among the backend and even if the load balancer server or the backend server were failing, the system was still able to keep working. Several test was also proofing that using a load balancing will decrease response time and increase the system throughput which causes an increase in the performance of the system. Based on this research, using Nginx as a load balancing combined with failover can be used as one of the solution to overcome an increasing traffic in a website.

Keywords: load balancing, network, nginx, keepalived.

1. PENDAHULUAN

Seiring dengan perkembangan teknologi, jumlah penggunaan layanan web semakin meningkat. Hal ini menyebabkan situs-situs web popular memiliki jumlah *traffic* yang tinggi. Kegiatan atau acara tertentu juga dapat menyebabkan naiknya jumlah *traffic* web suatu organisasi. Contohnya, pada masa pendaftaran masuk kuliah atau sekolah online, pada masa krs online, pada saat pemilu, ataupun event-event olahraga seperti piala dunia, olimpiade, dan event-event lain dapat menyebabkan peningkatan *traffic* web yang signifikan pada rentang waktu tersebut. Peningkatan jumlah *traffic* menyebabkan kerja server yang melayani permintaan menjadi semakin berat. Akibatnya performa server menurun

dan sering terjadi gangguan pada layanan-layanan web tersebut. Jika tidak ditangani, hal ini dapat mengakibatkan sistem ataupun situs web tersebut mati/down. Menurut Thamrin (2011), dalam jurnalnya yang berjudul perancangan *tools* berbasis python untuk memantau keaktifan server mengatakan bahwa faktor penting dalam sistem yang handal salah satunya adalah keaktifan server itu sendiri. Oleh karena itu, server yang mati dapat mengganggu layanan yang diberikannya.

Salah satu solusi untuk menangani masalah tersebut adalah dengan menggunakan sebuah server yang handal dengan performa yang tinggi. Solusi lain yang dapat diterapkan adalah teknologi *load balancing*. Ren, dkk (2012) berpendapat bahwa teknologi *load balancing* dapat membagi beban permintaan ke beberapa web server sehingga teknologi ini memiliki peranan penting dalam mencapai penggunaan sumber daya bersama yang efektif. Pandey, dkk (2015) mengatakan bahwa *load balancing* merupakan aspek penting yang dapat mendistribusikan beban kerja ke banyak server secara optimal yang menghasilkan waktu tanggap yang baik dan meningkatkan tingkat kepuasaan pengguna, meningkatkan efisiensi penggunaan sumber daya, dan berdampak pada peningkatan performa secara keseluruhan. Menurut Friedrich, dkk. (2012), *load balancing* diperlukan untuk efisiensi kegunaan sumber daya komputer dalam sistem paralalel dan terdistribusi. Tujuannya untuk merelokasi beban sehingga nantinya tiap *node* menerima jumlah beban yang sama. Selain itu, Yang (2016) menyatakan tujuan utama *load balancing* adalah untuk membagikan beban secara adil ke seluruh *nodes* yang ada. Ia menerapkan algoritma dinamis ke dalam rancangannya sehingga *load balancing* dapat merubah algoritma sesuai dengan kondisi yang ada.

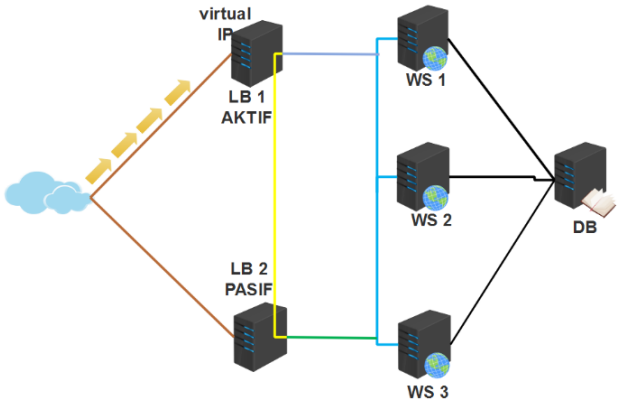
Hartomo (2015) melakukan implementasi web server *load balancing* pada mesin virtual. Hasilnya menunjukkan bahwa teknik *load balance* bekerja dengan cara membagi beban yang diterima oleh server dan server lain yang aktif, mampu menggantikan peran server yang mati. Penelitiannya juga turut membuktikan hasil penggunaan dua software *load balancing* pound dan haproxy dimana keduanya sama-sama mampu meningkatkan kemampuan server namun terbukti haproxy sedikit lebih unggul.

Load balancing dapat mengatasi kegagalan yang terjadi pada sisi web server. Akan tetapi, jika *load balancing* mengalami kegagalan, seluruh layanan dapat terhenti. Oleh karena itu, diperlukan *failover* pada *load balancing* yang terpasang. Noviyanto, dkk (2015) mendefinisikan bahwa *failover* merupakan kemampuan sebuah sistem untuk berpindah secara manual atau otomatis jika salah satu sistem mengalami kegagalan sehingga sistem lain menjadi backup bagi sistem yang mengalami kegagalan. Cara ini memungkinkan sistem dapat terus berjalan walaupun sistem utama mengalami kegagalan, selama sistem cadangan tetap tersedia.

Penelitian ini bertujuan untuk mengimplementasikan *load balancing* pada web server agar dapat meningkatkan kinerja sistem, serta mampu mengantisipasi kegagalan sistem melalui teknik *failover*. Manfaat dari penelitian ini adalah memahami konsep *load balancing* yang dipadu dengan teknik *failover* dengan menggunakan Nginx dan KeepAlived pada server berbasis Ubuntu.

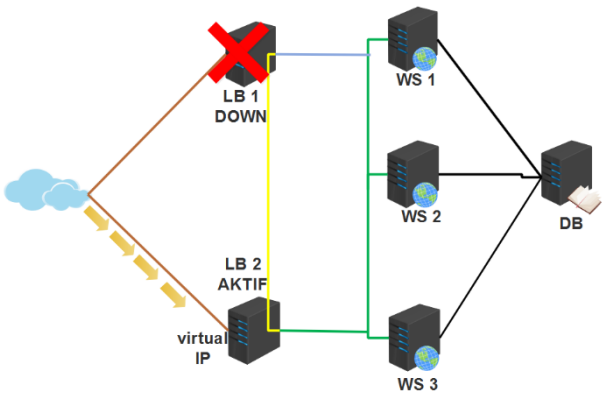
2. METODE

Penelitian ini dilakukan secara eksperimental di Laboratorium Jaringan Komputer Fakultas Komunikasi dan Informatika, Universitas Muhammadiyah Surakarta. Rancangan sistem *load balancing* dalam penelitian ini menggunakan 6 buah komputer. Dari 6 komputer tersebut, 2 diantaranya berperan sebagai *load balancing*, 3 berperan sebagai web server, dan 1 berperan sebagai server *database*. Gambaran umum dari rancangan sistem *load balancing* dapat dilihat pada gambar 1.



Gambar 1. Sistem *load balancing* utama

Gambar 1 menunjukkan bahwa sistem *load balancing* menggunakan metode aktif-pasif. Dalam metode aktif-pasif, hanya satu *load balancing* aktif/master yang akan melayani permintaan yang masuk. *Load balancing* lain atau *load balancing* pasif/slave hanya ikut memonitor kondisi frontend dan backend server. Apabila terjadi kegagalan pada *load balancing* aktif, sistem akan melakukan *failover* sehingga *load balancing* pasif akan mengambil alih status aktif dari master sementara waktu hingga master berfungsi kembali. Gambar 2 menunjukkan apa yang terjadi setelah *failover*.



Gambar 2. Sistem *load balancing* saat *failover*

Apabila sistem telah diimplementasikan, maka sistem siap diuji dalam beberapa skenario yang telah disediakan.

2.1. ALAT DAN BAHAN

Tahapan ini menentukan hal-hal yang dibutuhkan dalam proses penelitian seperti hardware dan software. Terdapat 7 komputer yang digunakan dalam penelitian ini, 6 sebagai server, dan 1 sebagai klien. Ke-6 komputer yang digunakan sebagai server tersebut memiliki spesifikasi hardware komputer sebagai berikut: Intel Core i3-2120 3.30GHz, 4GB DDR3, 500GB HDD dengan sistem operasi Ubuntu 12.10. Sedangkan komputer klien memiliki spesifikasi hardware sebagai berikut: Intel Core i3-2348 2.30GHz, 6GB DDR3, 500GB HDD dengan sistem operasi Windows 10. Untuk software yang digunakan, antara lain nginx, keepalived, jmeter, apache2, mysql, htop, saidar, nano, dan putty.

2.2. IMPLEMENTASI SISTEM

a) Pemberian IP address

Proses konfigurasi dimulai dengan pengaturan ip address tiap-tiap komputer. Pada *load balancing*, IP address yang digunakan adalah 192.168.52.9/10. Sedangkan pada web server akan diberikan IP 192.168.52.2/6/12. Untuk server *database*, akan diberikan IP 192.168.52.16. Karena terdapat 2 *load balancing*, diberikan 1 virtual IP yang dapat berpindah antar *load balancing*. Virtual IP ini berfungsi sebagai IP dari situs web/aplikasi web yang digunakan. IP address untuk virtual IP adalah 192.168.52.99. Pengaturan IP address pada Ubuntu dapat dilakukan dengan cara mengubah isi file $/etc/network/interface. Hasil perubahan dapat dilihat dengan menggunakan perintah $ifconfig.

b) Konfigurasi *Load balancing*

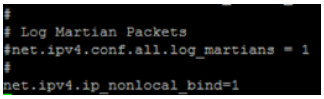
Pada kedua server *load balancing*, perlu dilakukan instalasi nginx. Nginx merupakan aplikasi yang dapat bertindak sebagai web server dan reverse proxy atau *load balance*r. Perintah instalasi nginx melalui repository Ubuntu adalah $apt-get install nginx. Lalu dilakukan penambahkan file konfigurasi baru dengan perintah $sudo nano /etc/nginx/conf.d/lb.conf. Restart nginx agar perubahan diterapkan. File konfigurasi tersebut memiliki isi seperti pada gambar 3.

```
upstream myapp1 {
        #least_conn;
        server 192.168.52.2:80 max_fails=3 fail_timeout=30s;
        server 192.168.52.6:80 max_fails=3 fail_timeout=30s;
        server 192.168.52.12:80 max_fails=3 fail_timeout=30s;
        }
    server {
        listen 192.168.52.99:80;
        access_log /var/log/nginx/lb.access.log;
        error_log  /var/log/nginx/lb-error.log error;

        location / {
            proxy_pass http://myapp1;

        }
}
```

Gambar 3. Konfigurasi lb.conf

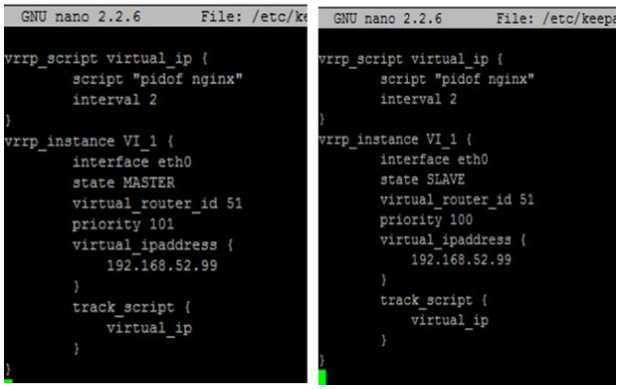
c) Konfigurasi *Failover*

Konfigurasi *failover* dimulai dengan menambahkan slot untuk ip virtual pada kedua server *load balancing*. Caranya dengan menambahkan baris net.ipv4.ip_nonlocal_bind=1 pada file $/etc/sysctl.conf. Agar perubahan diterapkan, masukkan perintah $sudo sysctl –p.



Gambar 4. Penambahan slot ip.

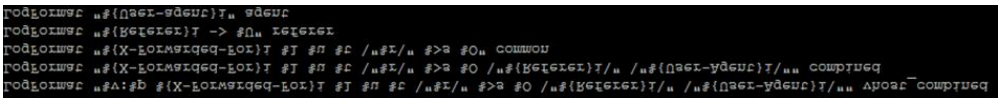
Selanjutnya, instalasi keepalived dengan perintah $sudo apt-get install keepalived. Gunakan perintah $sudo nano /etc/keepalived/keepalived.conf untuk membuat file konfigurasi baru. Simpan konifgurasi tersebut, lalu restart keepalived. Terdapat perbedaan pada kedua file konfigurasi KeepAlived di kedua *load balancer*. Perbedaan tersebut ada pada angka priority. Priority pada server *load balancing* master harus lebih tinggi dibandingkan pada server *load balancing* slave, tujuannya adalah agar kedua *load balancing* tidak memiliki ip virtual secara bersamaan.



Gambar 5. Konfigurasi keepalived.conf

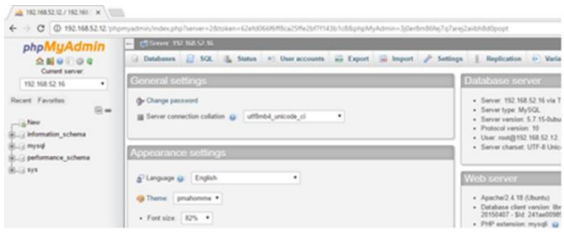
d) Konfigurasi server backend dan server *database*

Server backend dalam penelitian ini menggunakan apache2, php5, dan phpmyadmin. Agar backend dapat melihat ip klien, edit kata %h menjadi %{X-Forwarded-For}i pada file konfigurasi $/etc/apache2/apache2.conf.



Gambar 6. Konfigurasi file apache2.conf

Pada server *database* software yang digunakan adalah mysql. Selain mudah, mysql dapat di akses secara remote melalui phpmyadmin.

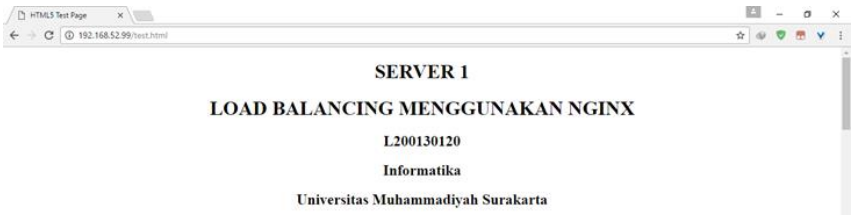


Gambar 7. Akses server *database* menggunakan phpmyadmin.

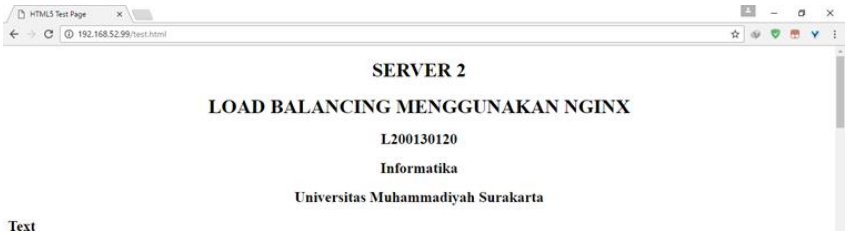
2.3 PENGUJIAN SISTEM

Pada tahapan ini, sistem diuji menggunakan aplikasi jmeter untuk melakukan simulasi pemberian beban dari komputer yang bertindak sebagai klien. Saat melakukan pengujian, peneliti memonitoring keadaan server-server terutama server backend untuk mengetahui performa dari server backend dengan bantuan beberapa software monitoring seperti saidar, htop, top, dan juga beberapa perintah dasar pada linux seperti tail, awk, grep. Hasil-hasil dari uji coba tersebut akan dimuat dalam laporan beserta analisis dari peneliti.

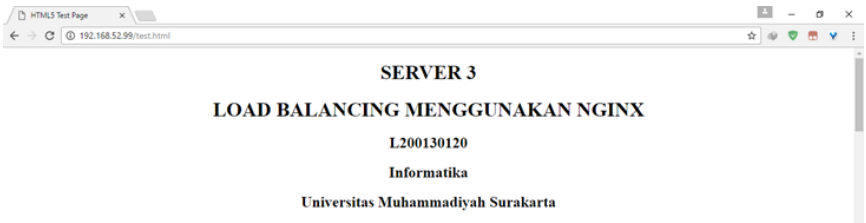
Hasil Pemasangan *load balancing* ini dapat diakses melalui browser dengan alamat ip virtual yang terpasang yaitu http://192.168.52.99/test.html. Untuk melihat server mana yang sedang melayani user, maka pada sisi backend dapat dipasang konten berbeda.



Gambar 8. Server 1 melayani pengguna



Gambar 9. Server 2 melayani pengguna



Gambar 10. Server 3 melayani pengguna

3. HASIL, PEMBAHASAN, DAN DISKUSI

Pengujian sistem *load balancing* yang terpasang dilakukan dengan tiga tahap berbeda dengan beberapa macam skenario. Tabel berikut akan memberikan gambaran pengujian yang telah dilakukan.

Tabel 1. Tabel Pengujian

No	Skenario uji	Parameter	Hasil yang diharapkan	Hasil
1	Pembagian beban	Jumlah akses	Semua Backend server menerima jumlah akses yang sama	sesuai
2	Salah satu backend mengalami kegagalan	Jumlah *Request* terlayani	Sistem tetap berjalan dan semua/sebagian besar *request* dilayani oleh sistem	Sesuai
3	Server utama *load balancing* mengalami kegagalan	Jumlah *Request* terlayani	Sistem tetap berjalan dan semua/sebagian besar *request* dilayani oleh sistem	Sesuai
4	Pengujian akses laman html	Waktu response	Sistem mampu melayani *request* lebih cepat	Tidak sesuai
		Throughput	Sistem mampu melayani lebih banyak *request* dalam tiap detik	Tidak sesuai
		error	Tidak ada *request* yang gagal	Sesuai
5	Pengujian akses laman php untuk menghitung phi	Waktu response	Sistem mampu melayani *request* lebih cepat	Sesuai
		Throughput	Sistem mampu melayani lebih banyak *request* dalam tiap detik	Sesuai
		error	Tidak ada *request* yang gagal	Sesuai
6	Pengujian input data dengan php	Waktu response	Sistem memiliki waktu response lebih cepat	Sesuai
		Throughput	Sistem mampu melayani lebih banyak *request* dalam tiap detik	Sesuai

		Jumlah data	Data yang masuk dalam database sesuai dengan jumlah *request* yang dilayani	Sesuai
		error	Tidak ada *request* yang gagal	Sesuai
7	Pengujian memasukkan gambar ke dalam database	Jumlah *request*	Sistem mampu melayani jumlah *request* yang cukup besar	Sesuai
		Jumlah data	Data yang masuk dalam database sesuai dengan *request* yang berhasil dilayani	Sesuai
		Error	Error terjadi dari sisi database	Sesuai

1) Pengujian Pembagian Beban

Pengujian ini dilakukan dengan cara mensimulasikan sejumlah klien yang mengakses sistem *load balancing* dimana setiap satu klien melakukan sekali *request*. Semua *request* tersebut, direkam oleh log apache pada backend-backend server. Data-data pada log apache tersebut dihitung dengan menggunakan perintah grep. Hasilnya dapat dilihat pada tabel berikut.

Tabel 2. Pembagian Beban

No	***Total request***	**Server 1**	**Server 2**	**Server 3**
1	10	3	4	3
2	50	17	16	17

Sebelum setiap pengujian, isi log apache pada setiap backend server dihapus. Berdasarkan tabel 2, pada pengujian pertama menggunakan 10 *request*, urutan pelayanan *request* dimulai dari server 2, server 3, server 1 dan terus berulang sampai semua *request* terlayani. Hasilnya menunjukkan bahwa server 1 melayani 3 *request*, server 2 melayani 4 *request*, sedangkan server 3 melayani 3 *request*. Pada uji coba kedua dengan 50 *request*, urutan pelayanan *request* adalah server 3, server 1, server 2. Dimana server 1 melayani 17 *request*, server 2 melayani 16 *request* dan server 3 melayani 17 *request*. Dari hasil tersebut terlihat bahwa *load balancer* membagikan beban secara merata secara berurutan sesuai dengan algoritma round robin.

2) Pengujian Ketersediaan Layanan

Pengujian selanjutnya yaitu dengan cara mematikan salah satu backend server yang ada. Pengujian ini dilakukan dengan memberikan 10 *request* kepada sistem yang sebelum pengujian dilakukan, satu atau dua backend server dimatikan. Tujuannya adalah untuk melihat apakah akan terjadi gangguan pada pelayanan oleh backend server.

Tabel 3. Ketersediaan saat terjadi kegagalan pada sisi backend

No Uji	*Total request*	Server 1	Server 2	Server 3
1	10	5	5	down
2	10	5	down	5
3	10	Down	down	10

Berdasarkan tabel 3, pada pengujian pertama, server 3 dalam keadaan mati sehingga semua *request* dilayani oleh server 1 dan server 2. Pada pengujian kedua, server 2 dalam keadaan mati sehingga semua *request* dilayani oleh server 1 dan server 3. Pada pengujian ketiga, server 1 dan server 2 dalam keadaan mati sehingga semua *request* dilayani oleh server 3. Terbukti bahwa apabila salah satu backend server mengalami kegagalan, hal ini tidak akan mengganggu sistem yang sedang berjalan sehingga pengguna tetap mendapatkan pelayanan dari sistem.

Pengujian selanjutnya dilakukan untuk menguji *failover* pada *load balancing* sistem. Pengujian ini dilakukan dengan cara memberikan beban sebanyak 100 *request* dengan ketentuan 1 *request* perdetik selama 100 detik dimana pada pertengahan uji, server *load balancer* yang aktif akan sengaja dimatikan dan memaksa terjadinya *failover* ke server *load balancer* pasif. Berikut hasil percobaannya.

Tabel 3. Hasil pengujian *failover* pada *load balancer*

No	*Request*	Berhasil	*Error*	Keterangan
1	100	98	2	LB 1 ke LB 2
2	100	100	0	LB 2 ke LB 1

Uji coba pertama dari tabel 3 dilakukan dengan cara mematikan server *load balancer* 1 saat sedang dilakukan pemberian *request*. Hasilnya menunjukkan bahwa ketika sedang terjadi perpindahan sistem dari server *load balancer* 1 ke server *load balancer* 2, *request* tidak dapat masuk karena sistem sedang offline. Pada percobaan pertama, sistem memerlukan waktu sektiar 5 detik untuk berpindah dari *load balancer* 1 ke *load balancer* 2 dan selama perpindahan tersebut terdapat 2 *request* oleh user yang tidak dilayani. Pada pengujian kedua, perpindahan sistem terjadi dari server *load balancer* 2 ke server *load balancer* 1. Pada pengujian ini, tidak terjadi *error* sama sekali yang menujukkan bahwa perpindahan sistem dari sistem cadangan kembali ke sistem utama tidak akan mempengaruhi jalannya sistem web. Pengujian ini membuktikan bahwa sistem *load balancing* yang terpasang mampu melakukan *failover* untuk mengantisipasi kegagalan pada server *load balancer*.

3) Pengujian Kinerja Server

Pengujian Kinerja Server merupakan pengujian untuk mengamati performa backend server dalam keadaan diberi beban yang cukup banyak. Tujuan dari pengujian ini adalah untuk melihat

perbandingan performa setelah menggunakan *load balancing* dengan sebelum menggunakan pada load balancing. Performa dihitung berdasarkan waktu respon yaitu waktu yang diperlukan sistem untuk melayani *request* dan juga throughput yaitu banyaknya request yang dapat dilayani sistem tiap detiknya. Jmeter akan merekam kedua hasil tersebut.

a) Pengujian dengan laman html

Pada pengujian ini, dilakukan dengan cara memberikan sejumlah *request* pada sistem web yang didalamnya telah dipasang laman web html seberat 12 kB. Pengujian ini dilakukan dengan cara membandingkan hasil saat menggunakan *load balancing* dan hasil saat menggunakan server web tunggal. Berikut adalah hasil dari pengujian ini.

Tabel 5. Hasil uji dengan laman html

No	User	*Total request*	Sistem yang digunakan	Waktu respon rata-rata (ms)	*Throughput* (*request*/s)	*error*
1	400	10000	Tunggal	3	330,5	0
			LB	4	331,6	0
2	800	20000	Tunggal	3	657,1	0
			LB	7	658,3	0
3	1200	30000	Tunggal	93	907,0	0
			LB	239	805,7	0

Berdasarkan tabel 5, pengujian pertama menggunakan 400 *concurrent* user dengan total 10000 *request* yang diberikan dalam kurun waktu 30 detik. Terlihat bahwa rata-rata respone time saat menggunakan server tunggal adalah 3 ms untuk melayani setiap *request* masuk. Hasil ini lebih baik jika dibanding dengan penggunaan *load balancing* yang mencapai 4 ms untuk tiap *request*-nya. Sedangkan *throughput* tidak banyak perbedaan diantara keduanya yaitu berkisar pada angka 331 *request* per detik. Pada pengujian kedua, dengan menggunakan 800 *concurrent* user dengan *Total request* sebanyak 20000 yang diberikan dalam kurun waktu 30 detik, terlihat bahwa waktu rata-rata *request* dilayani oleh server tunggal adalah 3 ms sedangkan untuk *load balancing* membutuhkan waktu 7 ms. Untuk *throughput* keduanya memiliki jumlah yang relative sama yaitu sektiar 657 *request* per detik. Pada pengujian ketiga dengan 1200 *concurrent* user dan *Total request* sebanyak 30000 yang diberikan dalam kurun waktu 30 detik, menghasilkan waktu respon sebesar 93 ms pada server tunggal. Pada saat menggunakan *load balancing*, waktu respon yang didapatkan tidak lebih baik yaitu sebesar 239 ms. Untuk *throughput* pada server tunggal juga lebih baik yaitu sebesar 907 *request* per detik jika dibandingkan dengan *load balancing* yang sebesar 805,7 *request* per detik. Dalam pengujian ini, penggunaan *load balancing* pada sistem terbukti tidak memberikan dampak lebih baik.

b) Pengujian dengan laman php

Pengujian ini dilakukan dengan memberikan sejumlah *request* kepada sistem web yang didalamnya telah terpasang sebuah file php. File php tersebut akan melakukan perhitungan untuk mencari nilai phi disisi server dan hasilnya akan dikirim ke user sebagai respon dari permintaan yang dilakukan user. Pengujian ini membandingkan antara hasil dengan menggunakan sistem *load balancing* dan hasil menggunakan server web tunggal. Berikut ini merupakan hasil dari pengujian-pengujian tersebut.

Tabel 6. Hasil uji dengan laman php

No	User	*Total request*	Sistem yang digunakan	Waktu respon rata-rata (ms)	*Throughput* (*request*/s)	*Error* (%)
1	400	10000	Tunggal	356	256,8	0
			LB	14	327,9	0
2	800	20000	Tunggal	1525	261,0	1,01
			LB	17	651,5	0
3	1200	30000	Tunggal	2840	218,1	4,80
			LB	349	760,0	0

Berdasarkan tabel 6, pengujian pertama menggunakan 400 *concurrent* user dengan total 10000 *request* yang dikirim dalam kurun waktu 30 detik. Waktu respon rata-rata sistem untuk melayani *request* yang masuk saat menggunakan server tunggal adalah sebesar 356 ms. Sedangkan waktu respon rata-rata saat menggunakan *load balancing* menunjukkan hasil yang lebih baik yaitu sebesar 14 ms. *Throughput* saat menggunakan server tunggal hanya mencapai angka 256,8 *request* per detik jika dibanding dengan saat menggunakan *load balancing* yang sebesar 327,9 *request* per detik. Pada pengujian kedua dengan menggunakan 800 *concurrent* user yang melakukan *Total request* sebanyak 20000 yang diberikan pada kurun waktu 30 detik, hasilnya menunjukkan waktu respon rata-rata pada server tunggal sebesar 1525 ms. Sedangkan saat menggunakan *load balancing*, waktu respon rata-rata sebesar 17 ms. *Throughput* pada server tunggal menunjukkan angka 261 *request* per detik sedangkan saat menggunakan *load balancing* hasilnya adalah 651,5 *request* per detik. Pada server tunggal 1,01% *request* dari *Total request* mengalami *error* sedangkan saat menggunakan *load balancing* tidak terdapat *request* yang mengalami *error*. Pengujian ketiga dengan menggunakan 1200 *concurrent* user dan *Total request* sebanyak 30000 yang diberikan dalam kurun waktu 30 detik menunjukkan bahwa waktu respon rata-rata saat menggunakan server tunggal adalah 2840 ms. Sedangkan waktu respon rata-rata saat menggunakan *load balancing* adalah 349 ms. *Throughput* pada server tunggal menunjukkan bahwa sistem mampu melayani 218,1 *request* per detik, sedangkan saat menggunakan *load balancing,* sistem mampu melayani 760 *request* per detik. Pengujian ketiga pada server tunggal menunjukkan bahwa terdapat 4,80% *request* dari *Total request* yang tidak terpenuhi sedangkan saat

menggunakan *load balancing*, tidak ada *request* yang gagal terpenuhi. Hasil pengujian-pengujian dengan laman php menunjukkan bahwa penggunaan sistem *load balancing* memiliki hasil lebih baik.

c) Pengujian dengan laman php dan *database*

Pengujian ini dilakukan dengan cara memberikan sejumlah *request* pada sistem web untuk melakukan input data kedalam *database* mysql melalui php. Dalam pengujian ini, data yang dimasukkan berupa 200 karakter string, timestamp, dan angka. Setiap *request* akan melakukan input data dua kali kedalam *database*. Hasil dari pengujian ini adalah sebagai berikut.

Tabel 7. Hasil uji input data dengan php

No	**User**	***Total request***	**Sistem yang digunakan**	**Waktu respon rata-rata (ms)**	***Throughput* (*request*/s)**	***Error* (%)**
1	400	10000	Tunggal	185	288,6	0
			LB	172	285,6	0
2	800	20000	Tunggal	247	571,6	0
			LB	187	576,7	0
3	1200	30000	Tunggal	477	658,6	0.32
			LB	196	836,9	0

Berdasarkan tabel 7, hasil dari pengujian pertama dengan menggunakan 400 *concurrent* user dengan *Total request* sebanyak 10000 yang diberikan dalam waktu 30 detik menunjukkan bahwa waktu respon rata-rata menggunakan server tunggal adalah 185 ms sedangkan waktu respon rata-rata saat menggunakan *load balancing* adalah 172 ms. *Throughput* yang didapat ketika menggunakan server tunggal adalah 288,6 *request* per detik dan saat menggunakan *load balancing* memiliki hasil 285,6 *request* per detik. Pada pengujian kedua dengan menggunakan 800 *concurrent* user dan juga *Total request* sebanyak 20000 yang diberikan dalam kurun waktu 30 detik menunjukkan bahwa waktu respon rata-rata saat menggunakan server tunggal adalah sebesar 247 ms sedangkan saat menggunakan *load balancing* waktu respon rata-rata sebesar 187 ms. *Throughput* pada server tunggal mencapai angka 571,6 *request* per detik sedangkan *throughput* saat menggunakan *load balancing* hanya sebesar 576,7 *request* per detik. Pada pengujian ketiga, menggunakan 1200 *concurrent* user dengan total 30000 *request* yang diberikan dalam kurun waktu 30 detik menunjukkan bahwa waktu respon rata-rata saat menggunakan server tunggal adalah 477 ms sedangkan saat menggunakan *load balancing* adalah 196 ms. *Throughput* pada server tunggal menunjukkan angka sebesar 658,6 *request*s/detik sedangkan saat menggunakan *load balancing* hasil *throughput* menunjukkan angka 836,9 *request* per detik. Pada pengujian ketiga ini, saat menggunakan server tunggal terdapat 0.32% *request* dari *Total request* yang mengalami kegagalan, sedangkan saat menggunakan *load balancing*, tidak terdapat *request* yang mengalami kegagalan. Pada pengujian ini, pemasangan *load balancing* sistem terbukti memberikan peningkatan kinerja.

Pada pengujian selanjutnya, dilakukan dengan cara memasukkan gambar kedalam *database* mysql. Gambar yang dimasukkan memiliki ekstensi jpg, namun pada mysql gambar akan memiliki format blob file. Ukuran gambar yang dimasukkan adalah sektiar 500 KB. Setiap *request* yang dilakukan oleh pengguna dalam pengujian kali ini akan memasukkan dua buah gambar. Perlu diketahui bahwa pengujian ini dilakukan setelah mysql di optimasi dengan cara memperbesar jumlah koneksi yang mampu dilayani mysql tiap waktu dan juga memperbesar ukuran log file untuk innoDB. Jumlah koneksi tersebut diperbesar dari aturan default sebesar 150 menjadi 500. Sedangkan ukuran log file diperbesar menjadi 128 MB. Hal ini dilakukan karena sebelum dioptimasi, pada sisi backend server sering mendapat peringatan *"to many connections"*, sedangkan pada sisi server *database* mendapat *error* dengan keterangan bahwa ukuran log files tidak cukup. Berikut merupakan hasil dari pengujian ini:

Tabel 8. Hasil uji input gambar ke mysql

No	User	*Total request*	Waktu Respon (ms)	*Throughput* (*request*/s)	*Error* (%)
1	100	2000	12302	6,8	0
2	200	4000	22241	8	20,42
3	300	6000	3619	62,5	89,32

Berdasarkan hasil yang terlihat pada tabel 8, pada pengujian pertama dengan 100 *concurrent* user dengan *Total request* sebanyak 2000 *request* yang diberikan dalam waktu 30 detik, waktu respon rata-rata tiap *request* yang diberikan oleh user adalah sektiar 12.302 ms dengan *throughput* sebanyak 6,8 *request* per detiknya. Pada pengujian kedua, dengan 200 *concurrent* user yang memberikan *Total request* sebanyak 4000 dalam waktu 30 detik menghasilkan waktu respon sebesar 22.241 ms dengan *throughput* sebesar 8 *request* perdetik. Dalam pengujian kedua, sektiar 20% *request* yang diberikan tidak/gagal dilayani oleh server. Pada pengujian ketiga dengan 300 *concurrent* user dengan *Total request* sebanyak 6000 dalam 30 detik memberikan hasil berupa waktu respon sebesar 3619 ms dengan *throughput* sebesar 62,5 *request* per detik. *Request* yang gagal dilayani pada percobaan ketiga mencapai 89% dari *Total request* yang diberikan. Pengujian input gambar ini membuktikan bahwa mysql tidak mampu melayani query berat dari 3 server backend didepannya walaupun telah dilakukan optimasi.

4. PENUTUP

Berdasarkan hasil pengujian dan pembahasan yang telah dilakukan dalam penelitian ini, dapat ditarik beberapa kesimpulan yaitu:

1. Sistem yang dirancang dapat membagikan beban secara merata ke beberapa backend server baik dalam keadaan semua server normal ataupun saat terjadi kegagalan pada salah satu backend server.
2. Sistem yang dirancang dapat meningkatkan ketersediaan karena mampu melakukan *failover* saat terjadi kegagalan baik di sisi *load balancer* ataupun di sisi backend server.
3. Berdasarkan pengujian menggunakan jmeter yang telah dilakukan, sistem tidak meningkatkan waktu respon dan *throughput* pada *request* terhadap laman html, akan tetapi saat melayani *request* laman php ataupun input data menggunakan php ke mysql, sistem *load balancing* mampu meningkatkan waktu respon dan *throughput* dengan baik.
4. Penggunaan *load balancing* membuat sisi *database* menjadi lebih berat karena hanya menggunakan satu buah *database* sehingga saat menjalankan *request* yang memerlukan query yang berat, banyak *request* yang gagal.

Saran bagi penelitian serupa yang akan datang adalah untuk dapat mengimplementasikan rancangan ke server dengan kondisi lingkungan yang nyata. Selain itu, penelitian selanjutnya disarankan untuk menggunakan lebih dari satu *database* untuk menopang sistem *load balancing* di depannya.

DAFTAR PUSTAKA

Thamrin, H. (2011). Perancangan Tools Berbasis Python Untuk Memantau Keaktifan Server. *KomuniTi,* 2(2), 25-31.

Ren, H., Lan, Y., & Yin, C. (2012). The Load Balancing Algorithm in Cloud Computing Environment. *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, PP, 925-928. doi: 10.1109/ICCSNT.2012.6526078

Pandey, S., Prassana, S., Kapil, S., & Rajeshwari, B. S. (2015). Load Balancing Techniques: A Comprehensive Study. *International Journal of Advance Research in Computer Science and Management Studies*, 3(4), 331-335.

Friedrich, T., Gairing, M., & Sauerwald, T. (2012). Quasirandom load balancing. *SIAM Journal on Computing*, 41(4), 747-771. doi:http://dx.doi.org/10.1137/100799216.

Yang, J. P., (2016). Elastic Load Balancing Using Self-Adaptive Replication Management. *IEEE Access*, PP(99), 1-10. doi: 10.1109/ACCESS.2016.2631490.

Hartomo, H. Y. (2015). Implementasi Web Server Load Balancing pada Mesin Virtual. *Skripsi*. Surakarta: Universitas Muhammadiyah Surakarta.

Noviyanto, A. B., Kumalasari, E., & Hamzah, A. (2015). Perancangan dan Impementasi Load Balancing Reverse Proxy Menggunakan HAProxy pada Aplikasi Web. *Jurnal Jaringan Komputer,* 3(1), 21-31.

Nginx Documentation. (2016). Using Nginx as Load Balancing. http://nginx.org/en/docs/http/load_balancing.html (diakses pada 25 November 2016)

Fletcher, A. (2002). LVS NAT + Keepalived HOWTO. http://www.keepalived.org/LVS-NAT-keepalived-HOWTO.html (diakses pada 10 Januari 2017).

Apache Jmeter Documentation. (2016). Component Reference. http://jmeter.apache.org/usermanual/component_reference.html#Aggregate_Report (diakses pada 30 November 2016)